**算法分析题**：5个物体，重量分别为3，5，7，8，9，价值分别为4，6，7，9，10，背包容量为22，物体不能分割，求装入背包物体最大价值，写出求解过程

解：

对于枚举的每个物品i，有状态转移方程：

m(i, j) = {

                max(m(i - 1, j), m(i - 1, j - wi) + vi)     j >= wi

// 选中的物品能装下

                m(i - 1, j)                                 j < wi

// 选中的物品装不下

           }

其中j为当前剩余空间，wi为物品i的重量，vi为物品i的价值

推导并绘制dp数组如下表所示

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i**  **j** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 17 | 20 | 21 | 22 |
|  |
| 1 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |  |
| 2 | 0 | 0 | 4 | 4 | 6 | 6 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |  |
| 3 | 0 | 0 | 4 | 4 | 6 | 6 | 7 | 10 | 10 | 11 | 11 | 13 | 13 | 13 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |  |
| 4 | 0 | 0 | 4 | 4 | 6 | 6 | 7 | 10 | 10 | 11 | 13 | 13 | 15 | 15 | 17 | 19 | 19 | 20 | 20 | 22 | 22 | 22 |  |
| 5 | 0 | 0 | 4 | 4 | 6 | 6 | 7 | 10 | 10 | 11 | 13 | 14 | 15 | 16 | 17 | 19 | 20 | 20 | 21 | 23 | 23 | 25 |  |

解得出当取第{2, 4, 5}个物体时，有最大价值25

**算法实现题：最少硬币问题**

/\*\*

 \* @param {number[]} T // 硬币面值

 \* @param {number[]} coins // 硬币个数

 \* @param {number} m // 找钱目标

 \* @return {number}

 \*/

const coinChange = (T, coins, m) => {

    // 二维数组作为备忘录消除重叠子问题

    let memo = new Array(m.length + 1).fill(0).map(\_ => new Array(coins.length + 1).fill(0))

    const dp = (T, coins, m) => {

        // 处理出口条件

        if (m < 0) return -1

        // m < 0 时问题不成立返回-1

        if (m === 0) return 0

        // m === 0 时无需解决，返回0

        if (memo[m][coins]) return memo[m][coins]

        // 查备忘录，避免重复计算

        let res = Number.MAX\_SAFE\_INTEGER

        // 将结果初始化为无穷大以判断是否有解

        for (let i = 0; i < T.length; i++) {

            // 枚举每一种硬币

            if (!coins[i]) continue

            // 如果所选的硬币已经用完，跳过

            coins[i]--

            // 该硬币个数-1

            let subRes = dp(T, coins, m - T[i])

            // 处理减去所选硬币面值和数量的子问题

            if (subRes === -1) continue

            // 如果子问题不成立则父问题无意义，跳过

            res = Math.min(res, subRes + 1)

            // 每次选取较小结果作为父问题结果

        }

        memo[m][coins] = (res === Number.MAX\_SAFE\_INTEGER) ? -1 : res

        // 枚举过所有硬币后若有解则将结果存入备忘录

        return memo[m][coins]

        // 返回结果

    }

    return dp(T, coins, m)

}

**3-4**

类比算法分析题，依次枚举每个物品i，但在计算时要考虑容积d

对于枚举的每个物品i，有状态转移方程：

m(i, j， k) =   {

        max(m(i - 1, j, k), m(i - 1, j - wi, k - di) + vi)     j >= wi

        // 选中的物品能装下

        m(i - 1, j， k)                                 j < wi

        // 选中的物品装不下

        }

其中j为当前剩余空间，wi为物品i的重量，di为物品i的体积，vi为物品i的价值

复杂度分析：

时间上，最坏情况下，需枚举每个物品选择与否的重量和体积并处理其子问题情况，共n \* w \* b次，每次执行所需要的时间为常数时间，故该算法的时间复杂度为O(n \* w \* b)

空间上，若使用备忘录减少重叠子问题，则需要大小为d \* n \* 2的二维数组进行状态转移，故空间复杂度为O(d \* n)